APPENDIX A -- 16/K Route Lookup Algorithm Pseudocode

```
t1_index = ip_addr[31:16]; // get the index to the T1_RIB
t1_entry[31:0] = T1_RIB[t1_index];
if (t1_entry[31] == 1'b0) {
    next_hop = t1_entry[15:6];
}
else {
    K = t1_entry[30:27] + 1; // get K value
    T2_RIB = t1_entry[26:0];
    t2_index = ip_addr[15: 16-K];

    //Load specific T2_RIB entry
    t2_entry[15:0] = T2_RIB[t2_index];

    next_hop = t2_entry[15:6];
}
```

APPENDIX B -- 16/K Route Update Algorithm Pseudocode

```
if (prefix_length <= 16) {
  prefix = ip_addr >> (32 - prefix_length);
  prefix_diff = 16 - prefix_length;  //how many bits shy of 16 is it?

  //Possibly multiple T1_RIB entries are matched, examine all T1_RIB
  //entries within applicable range and update if necessary.

  for (entry = 0; entry < 2^prefix_diff; entry++) {
    t1_index = (prefix << prefix_diff) + entry;
    if (T1_RIB[t1_index].mark_bit == 0) {
      old_prefix_length = GET_PREFIX_LENGTH(T1_RIB[t1_index]);
      if ( old_prefix_length <= prefix_length) {
        // set it with the new next hop and new prefix length
        T1_RIB[t1_index] = {next_hop,prefix_length};
      }
    } else { // the marker bit is 1

      /* There is at least one route with a prefix length of greater
       than 16 already.  Our entry matches all 2nd level entries.
       Change all entries that are less specific. */

      int L2_SIZE = 2^T1_RIB[t1_index].K;
      T2_RIB = GET_POINTER(T1_RIB[t1_index]);

      for (uj = 0; uj < L2_SIZE; uj++) {
        if ( T2_RIB[uj].prefix_length <= prefix_length) {
          /* assign the new next hop and new prefix length */
          T2_RIB[uj] = {next_hop,prefix_length};
        }
      }
    } // end of else
  } // end of for (entry = 0; ...)
} else { // the prefix length > 16
  new_K = prefix_length - 16;
  prefix = ip_addr[31:16];

  if ( T1_RIB[prefix].mark_bit == 0) { // the first bit is 0 in T1
    old_T1_entry = T1_RIB[prefix];

    // get a new table, put into T1 and mark the first bit to 1
    NEW_T2_RIB = New T2_RIB with 2^new_K entries
    T1_RIB[prefix] = {mark_bit=1,new_K,Pointer to NEW_T2_RIB};

    //Init table with old NH/PL data from T1_RIB
    for (uj = 0; uj < 2^new_K; uj++) {
          //populate new table with NH/PL data from T1
          NEW_T2_RIB[uj] = NH/PL data from old_T1_entry;
    }

    // Overlay single new NH/PL entry
    new_route_index = ip_addr[15: 16-new_K];
    T2_RIB[new_route_index] = {next_hop,prefix_length};
  } else { // the T1 mark bit is 1
    if (new_K > T1_RIB[prefix].old_K) {
       // We need a new larger L2 table.
       OLD_T2_RIB = T1_RIB[prefix].pointer;
```

```
            k_diff = new_K - T1_RIB[prefix].K;

            //Create new table and link into T1_RIB entry
            NEW_T2_RIB = New T2_RIB with 2^new_K entries
            T1_RIB[prefix] = {mark_bit=1,new_K,NEW_T2_RIB};

            //Populate new T2_RIB
            Copy old table to new table replicating each entry in old
            table into 2^k_diff entries in the new table.

            //overlay new route
            new_route_index = ip_addr[15: 16-new_K];
            T2_RIB[new_route_index] = {next_hop,prefix_length};

            //delete old table
            free(OLD_T2_RIB);
          } else {
            // Current L2 table is large enough - examine a subset of
            // current table for potential changes.
            T2_RIB = T1_RIB[prefix].pointer;
            k_diff = (T1_RIB[prefix].k - new_K);

            for (uj = 0; uj < 2^k_diff; uj++) {
              t2_index = (ip_addr[15: 16 - new_K] << k_diff) + uj;
              if ( T2_RIB[t2_index].prefix_length <= prefix_length) {
                  // update T2 with new NH and prefix length
                  T2_RIB[t2_index] = {next_hop,prefix_length};
              }
            }
          }
        }
      }
```

APPENDIX C -- 16/Kc Route Lookup Algorithm Pseudocode

```
t1_index = ip_addr[31:16]; // get the index to the T1_RIB
t1_entry[31:0] = T1_RIB[t1_index];

if (t1_entry[31] == 0) { // marker bit is 0
  next_hop = t1_entry[15:6];
}
else {
  //extract pointer to T2_RIB and K value.
  T2_Entry* p = t1_entry[26:0];
  K = t1_entry[30:27] + 1;

  if (K <= 6) {
    offset1 = 0;
    offset2 = ip_addr[ 15 : 16-K ];
  } else {
    offset1 = ip_addr[ 15 : 16- (K-6)];
    offset2 = ip_addr[ 16-(K-6)-1 : 16-K ];
  }
  bmp = p[offset1].bmp;
  nhpl_or_addr = p[offset1].nhpl_or_addr;
  all_ones = count_leading_ones(bmp, 63);
  leading_ones = count_leading_ones(bmp, offset2);

  if (all_ones <= 2) {
    if (leading_ones == 1)
          nhpl = nhpl_or_addr[63 : 48];  // get nhpl1
    else
          nhpl = nhpl_or_addr[47 : 32];   // get nhpl2
  } else { // nhpl_or_addr is an address
    nhpl = nhpl_or_addr[leading_ones - 1];
  }
  next_hop = GET_NEXT_HOP(nhpl);
}
```

APPENDIX D -- 16/Kc Route Update Algorithm Pseudocode

```
    if (prefix_length <= 16) {
       //change a subset of the L1 entries.
       foreach (matched T1_RIB entry) {
          if (T1_RIB[t1_index_of_match].mark_bit == 0) {
             [UPDATE_NHPL_DATA of T1_RIB entry if more specific]
          } else {
             //Entry matches entire T2_RIB
             [UPDATE_NHPL_ARRAY of entire T2_RIB for each less specific entry]
          }
       }
    } else { // end of if (prefix_length <= 16)
     //Only 1 matching T1 entry.  A subset of T2 needs modification
     t1_index_of_match = ip_addr[31:16];
     new_K = 16 - prefix_length;
     old_K = T1_RIB[t1_index_of_match].K;

     if (T1_RIB[t1_index_of_match].markbit==0) {
       [Create a new T2_RIB with MAX(1,2^(new_K-6)) entries]
       [Set bitmaps in T2_RIB to 0x8000000000000000LL]
       [Populate T2_RIB with T1_RIB's next_hop/prefix_length data]
       [Overlay new route into T2_RIB at appropriate position, changing the bitmap
and storing data into the nhpl array]
       //update T1_RIB to point to T2_RIB;
       T1_RIB[t1_index_of_match] = {mark_bit=1,new_K,T2_RIB base pointer};
     } else { // prefix_length > 16, mark bit == 1
       if (new_k <= old_K) {
             if (T1_RIB[t1_index_of_match].K <= 6) {
               //Only 1 T2_RIB entry.
               [Update a subset of T2_RIB_entry bitmap with new route where more
specific]
               [update nhpl where the new route is more specific]
             } else { //old_K > 6
               //Multiple (2^(old_K-6)) T2_RIB entries
               off1 = ip_addr[15:16-(old_K-6)];
               off2 = ip_addr[15-(old_K-6):10-(old_K-6)];

               if (new_K <= old_K - 6) {
                 //New route matches multiple entries in the T2_RIB
                 foreach (Matched entry in the T2_RIB) {
                    [Update nexthop/prefix_length array if this route is more
specific]
                    //No need to update bmp.
                 }
               } else {
                 // matched only 1 entry in T2_RIB
                [Update subset of T2_RIB_entry bitmap with new route where more
specific]
                [update subset of nhpl where the new route is more specific]
               }
             }
       } else { // new_K > old_K
             if (new_K <= 6) {
               // only 1 matched T2_RIB entry
               [Keep the same T2_RIB entry, but grow the bitmap]
```

```
            [overlay new entry into bitmap and next_hop/prefix_length array if
it is more specific]
            //update K value in T2_RIB entry
            T1_RIB[t1_index_of_match] = {mark_bit=1,new_K,T2_RIB};
          } else { //new_K > 6
            [Create a new T2_RIB with MAX(1,2^(new_K-6)) entries]

            if (new_K - old_K < 6) {
              // each bit is expanding into a fragment of a bmp.
              [Copy from old T2_RIB to new T2_RIB expanding each bit  into a
bitmap fragment of length 2^(new_K-old_K) bits]

            } else { //new_K - old_K >= 6
              // each bit is expanding into entire bmps.
              [Copy from old T2_RIB to new T2_RIB expanding each bit  into
2^(new_K-old_K-6) entries]
            }
            //update specific entry
            [Overlay new route into T2_RIB at appropriate position, changing
the bitmap and storing data into the nhpl array]
            //update t1 entry's k and ptr
            T1_RIB[t1_index_of_match] = {mark_bit=1,new_K,T2_RIB};
            //free old Table
            free(OLD_T2_RIB);
          }
      }
    }
  }
```